

# CS 1671 / CS 2071

## Human Language Technologies

Session 18: Transformers part 2

---

Michael Miller Yoder

October 22, 2026

# Assessments: project deliverable 2

- [Project deliverable 2](#) is due **next Thu Oct 29**
- Part 1: Basic data analysis (if any updates are required from the proposal)
- Part 2: Result from baseline approach
  - Ideally performance metric result from the baseline system you proposed
- Part 3: Advanced system (LLM) proposal
  - How might you use an LLM programmatically to attempt your task?
  - Zero-shot and at least 1 more advanced approach

# Assessments: project deliverable 2

Changes I made to instructions:

- Provide the text of the prompt/s you plan on using
- Discuss with me if you are **not** planning on testing a zero-shot decoder LLM
- Requiring an approach other than zero-shot prompting
  - Few-shot prompting
  - Chain-of-thought prompting
  - Fine-tuning if you have enough labeled data
- Peer review link is added
- Submit your code (GitHub link or files with a README)
- For groups using n-gram models, include informative features and error analysis

# SCI open LLMs available for the project

- Exact models available (feel free to use any)
  - gemma3 (gemma 3:27b)
  - llama3.1 (llama3.1:70b)
  - qwen3.8 (qwen3.8:27b)
- There are no rate limits. Just keep in mind it is a shared resource

# Assessments: quiz

Quiz during class **next Tue Oct 27** covering:

- Session 17 Transformers part 1
  - J+M 7-7.2
- Session 18 Transformers part 2
  - J+M 7.4-7.5, 7.9

# Overview: Transformers part 2

- Multi-headed attention
- Transformer blocks
- Transformer input and output details
  - Position embeddings
  - Language modeling head
- Activity: draw a diagram of a transformer-based LM

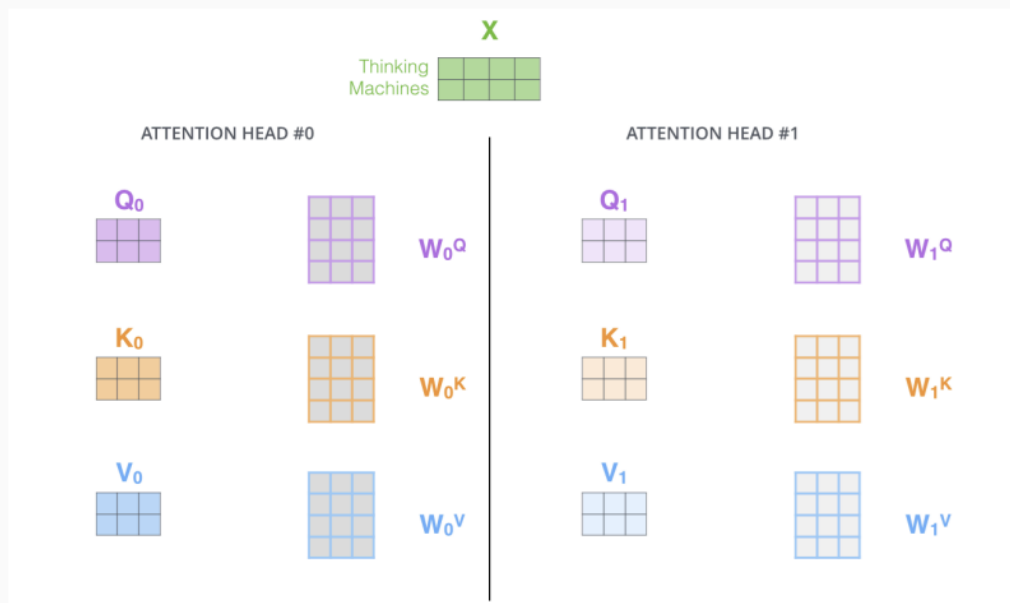
*Review:* Describe self-attention in transformers

# Multi-headed attention

---

# Multi-Headed Attention Expands Transformer Models' Ability to Focus on Different Positions

Maintain distinct weight matrices for each attention head—distinct representational subspaces:

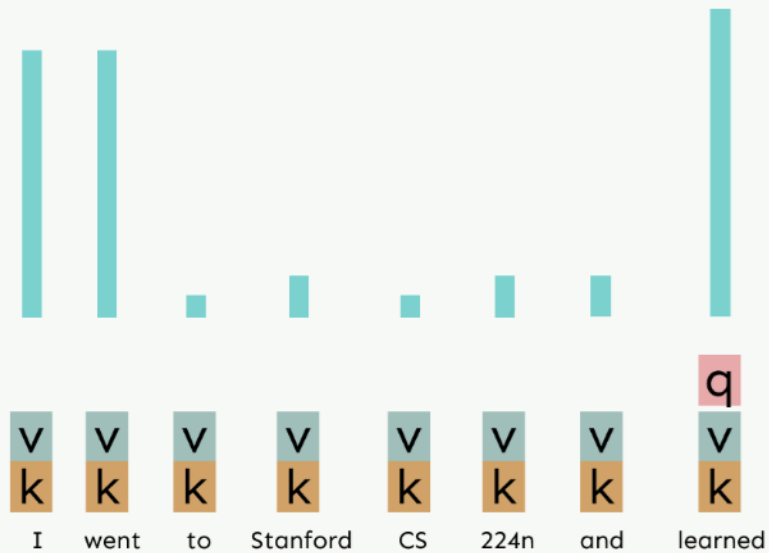


# Hypothetical example of multi-headed attention

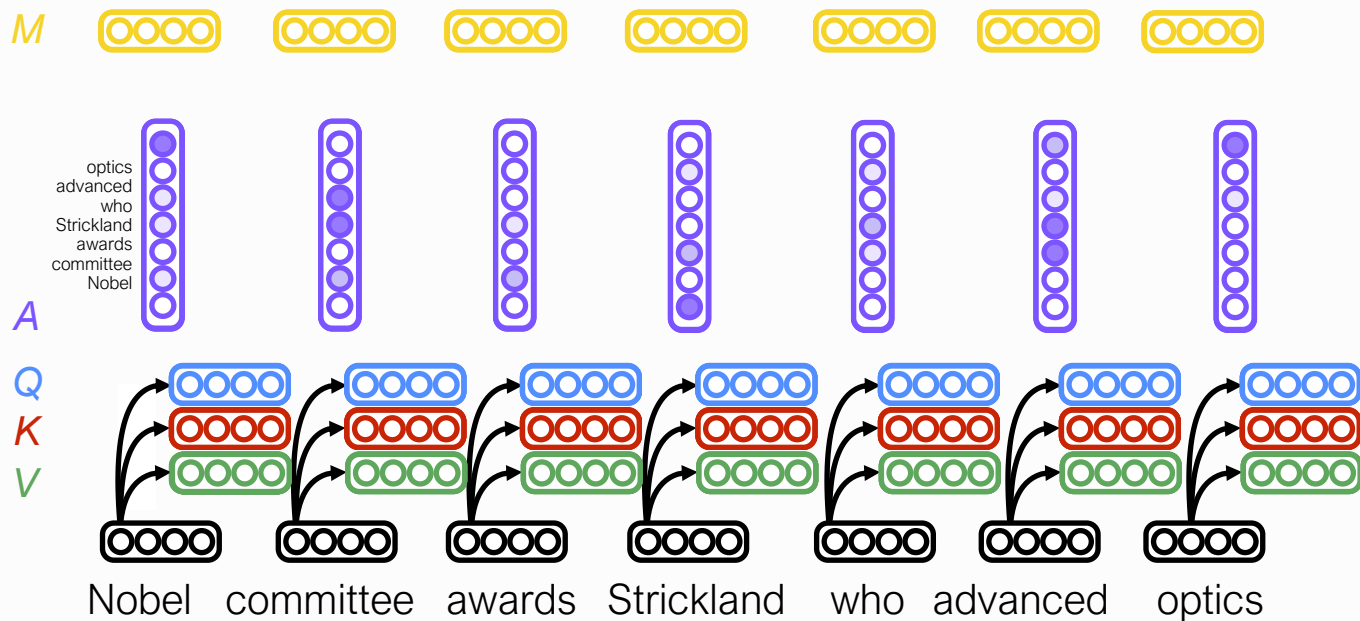
Attention head 1  
attends to entities



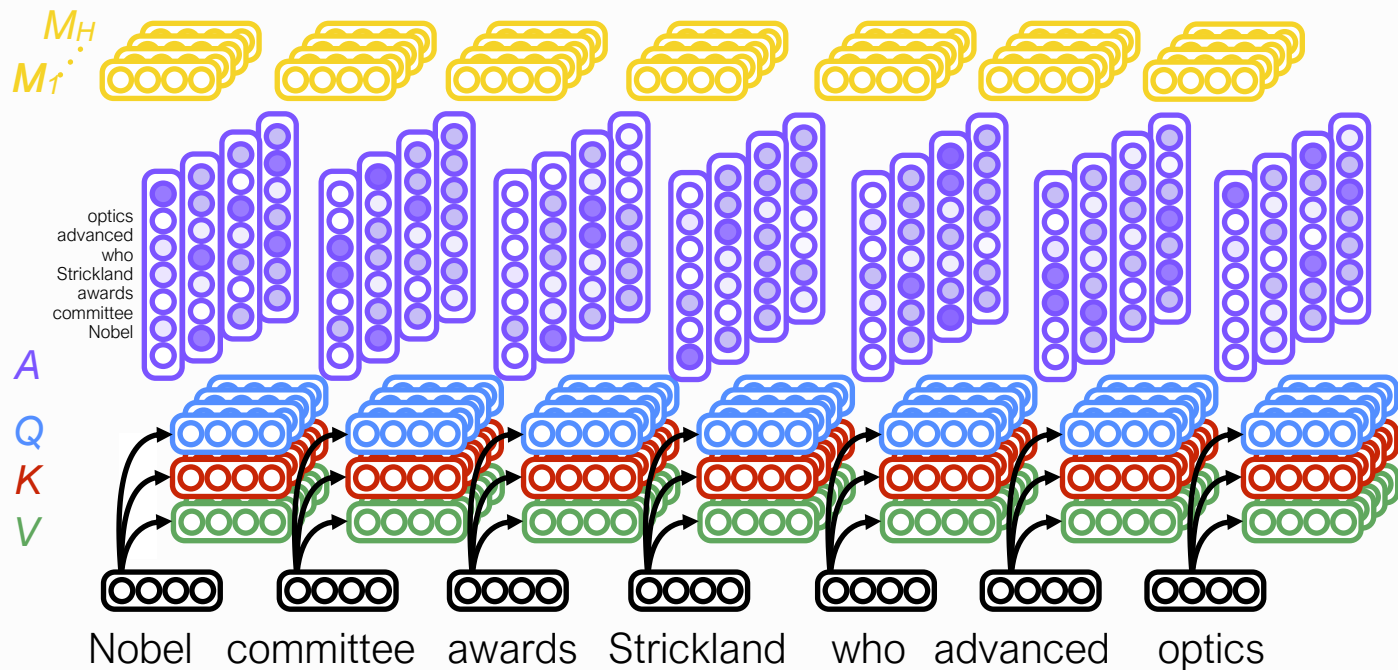
Attention head 2 attends to  
syntactically relevant words



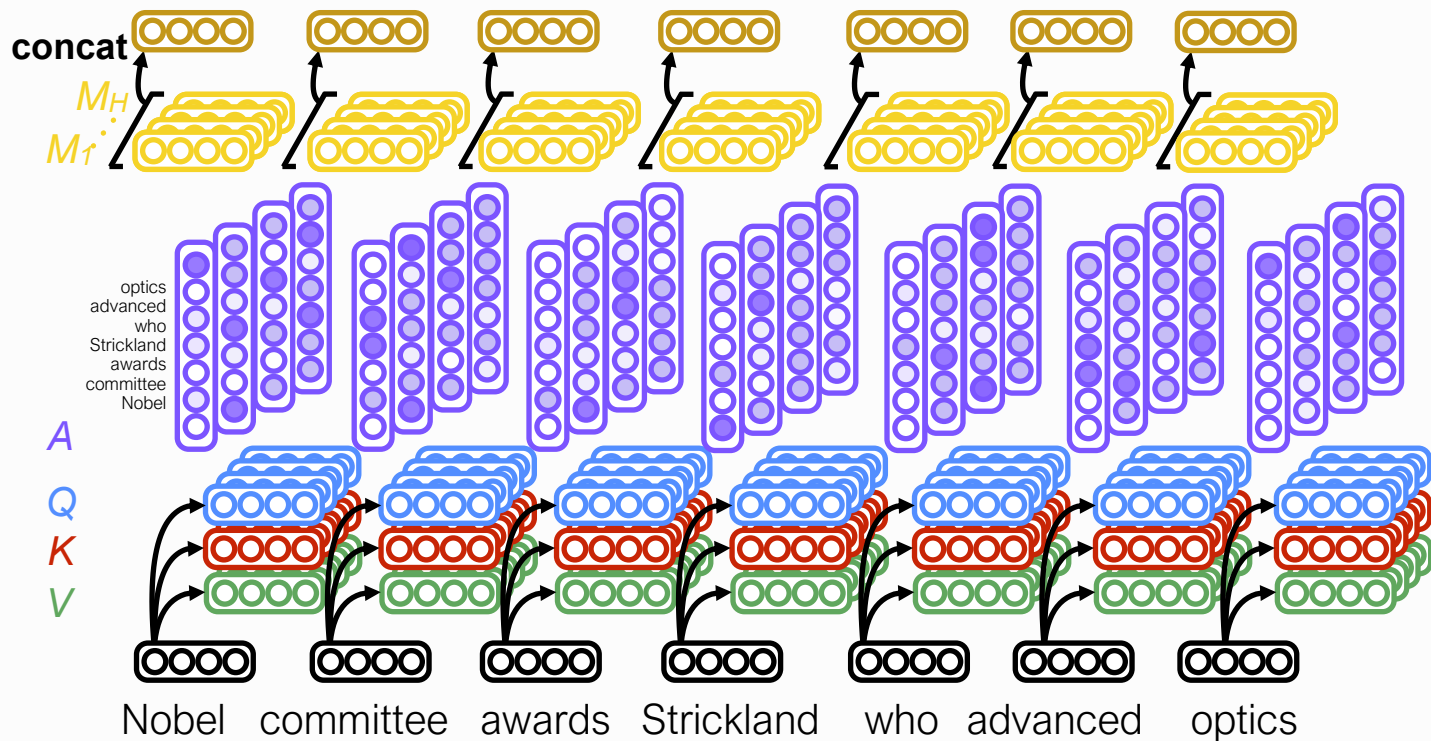
# Transformer self-attention



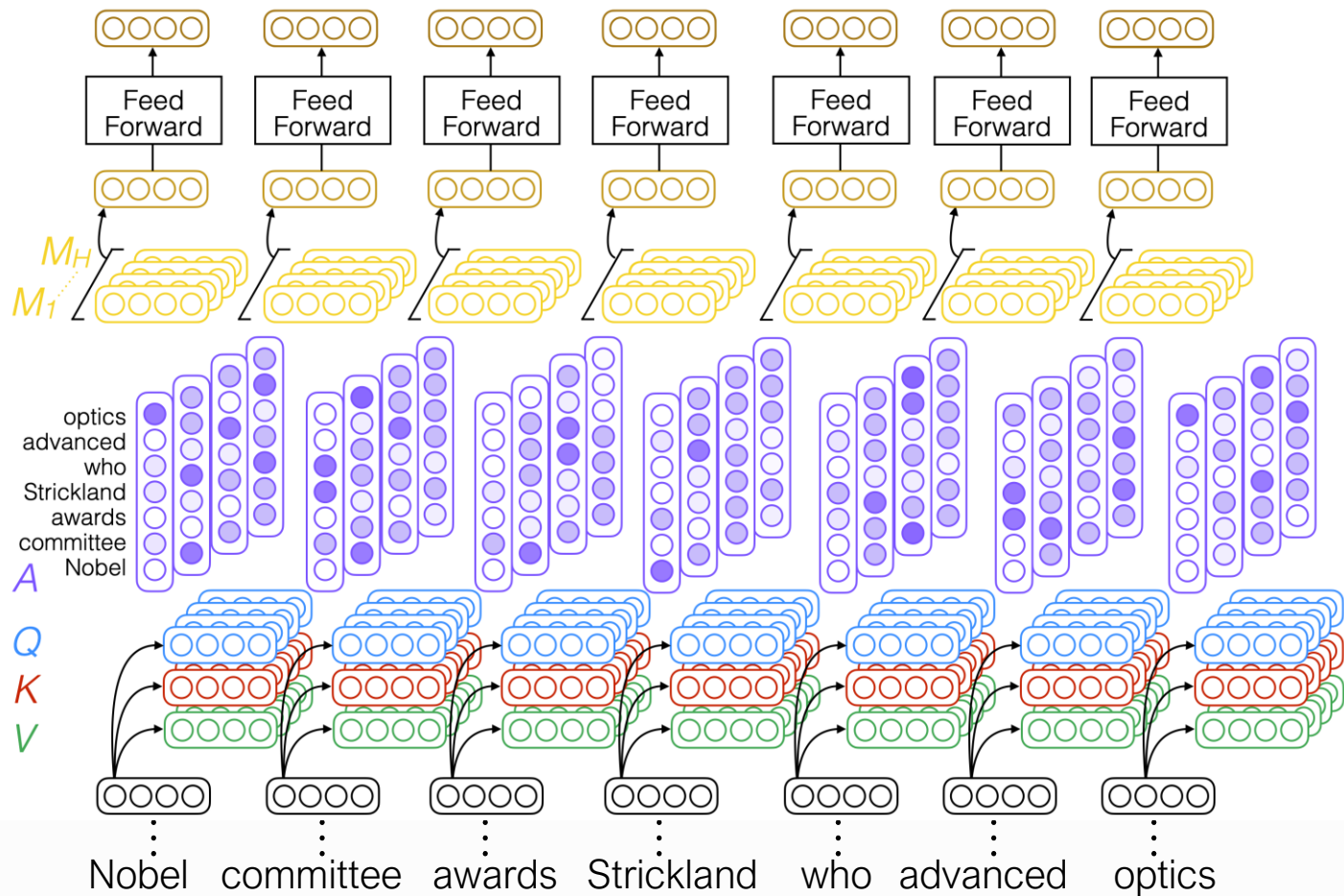
# Multi-head self-attention



# Multi-head self-attention



# Add a feedforward neural transformation for nonlinearity



# Transformer blocks

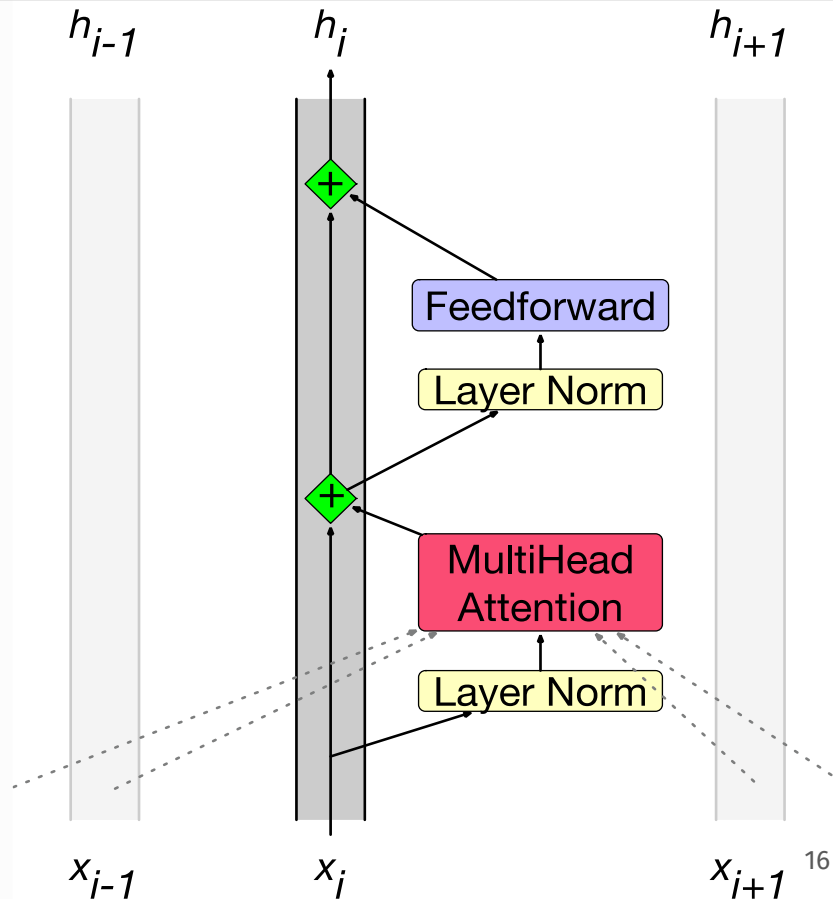
---

# Transformer blocks

Each block consists of:

- Self-attention
- Layer normalization and residual connections: tricks to optimize learning
- Feedforward neural network

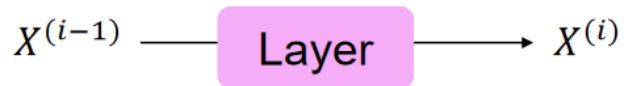
Output: 1 vector for every input token



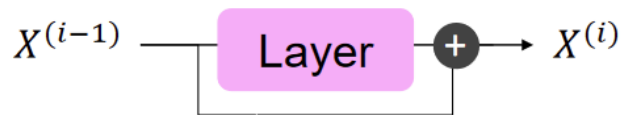
# Residual connections [He et al. 2016]

- **Residual connections** are a trick to help models train better.

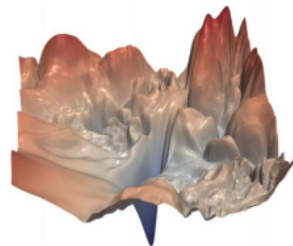
- Instead of  $X^{(i)} = \text{Layer}(X^{(i-1)})$  (where  $i$  represents the layer)



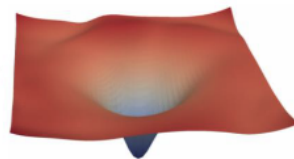
- We let  $X^{(i)} = X^{(i-1)} + \text{Layer}(X^{(i-1)})$  (so we only have to learn “the residual” from the previous layer)



- Gradient is **great** through the residual connection; it's 1!
- Bias towards the identity function!



[no residuals]



[residuals]

[Loss landscape visualization,  
[Li et al., 2018](#), on a ResNet]

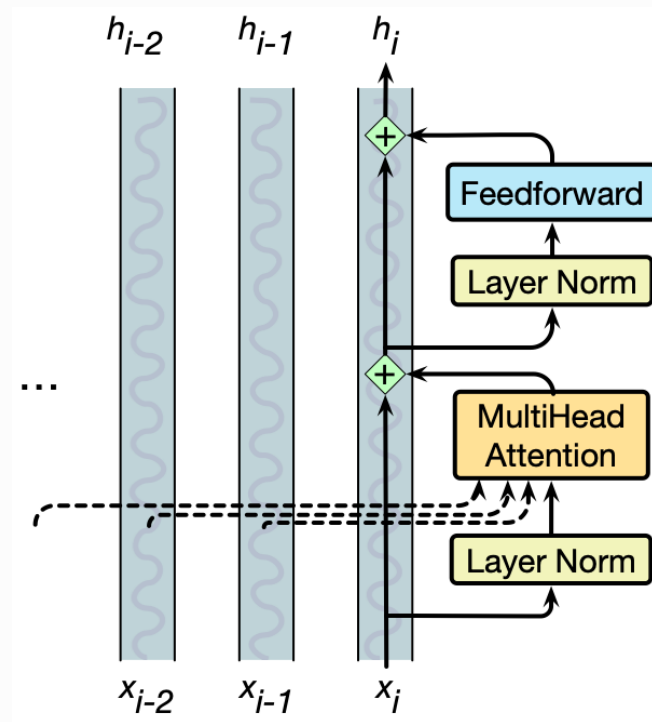
# Layer norm: the vector $\mathbf{x}_i$ is normalized twice

- Cuts down on uninformative variation to train faster
- Variation of the z-score from statistics, applied to each token's vector in a hidden layer
- Original implementation had a couple of other learnable parameters, but these can be omitted

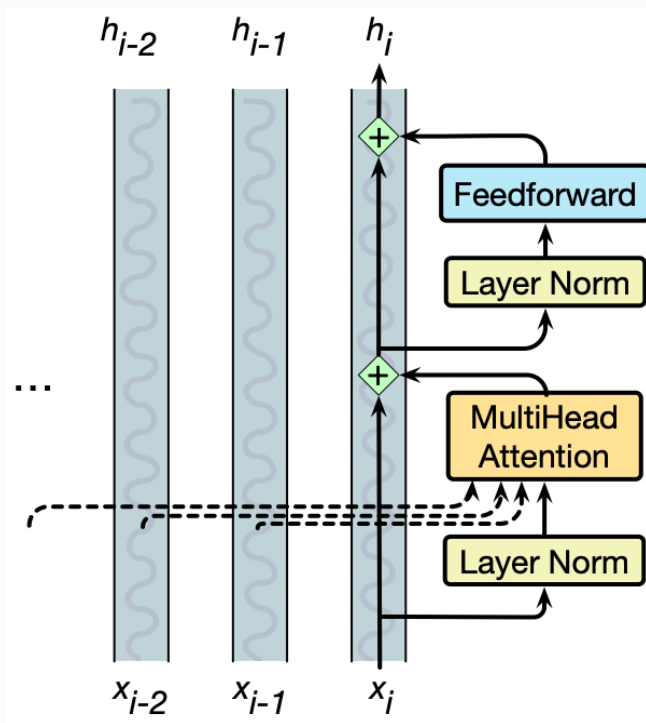
$$\hat{\mathbf{x}} = \frac{(\mathbf{x} - \mu)}{\sigma}$$

$$\mu = \frac{1}{d} \sum_{i=1}^d x_i$$

$$\sigma = \sqrt{\frac{1}{d} \sum_{i=1}^d (x_i - \mu)^2}$$



# Putting together a single transformer block

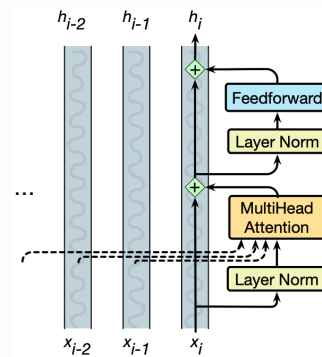


$$\begin{aligned}\mathbf{t}_i^1 &= \text{LayerNorm}(\mathbf{x}_i) \\ \mathbf{t}_i^2 &= \text{MultiHeadAttention}(\mathbf{t}_i^1, [\mathbf{x}_1^1, \dots, \mathbf{x}_N^1]) \\ \mathbf{t}_i^3 &= \mathbf{t}_i^2 + \mathbf{x}_i \\ \mathbf{t}_i^4 &= \text{LayerNorm}(\mathbf{t}_i^3) \\ \mathbf{t}_i^5 &= \text{FFN}(\mathbf{t}_i^4) \\ \mathbf{h}_i &= \mathbf{t}_i^5 + \mathbf{t}_i^3\end{aligned}$$

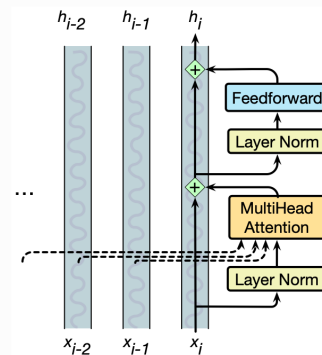
# A transformer is a stack of these blocks

- Vectors all have the same dimensionality  $d$
- $d$  is usually around 512 or 768
- Output of one transformer block becomes the input of another

Block 2



Block 1



- Transformer input and output

---

# Token and Position Embeddings

- The matrix  $X$  (of shape  $[N \times d]$ ) has an embedding for each word in the context.
  - $N$  is the total number of word instances (tokens)
  - $d$  is the dimension of word embeddings, a hyperparameter
- This embedding is created by adding two distinct embeddings for each input: **token** and **position** embeddings
- Since self-attention doesn't build in order information, we need to encode the order of the sentence in our keys, queries, and values

# Token Embeddings

Embedding matrix  $E$  has shape  $|V| \times d$

- One row for each of the  $|V|$  tokens in the vocabulary.
- Each word is a row vector of  $d$  dimensions

Given: string "*Thanks for all the*"

1. Tokenize with BPE and convert into vocab indices

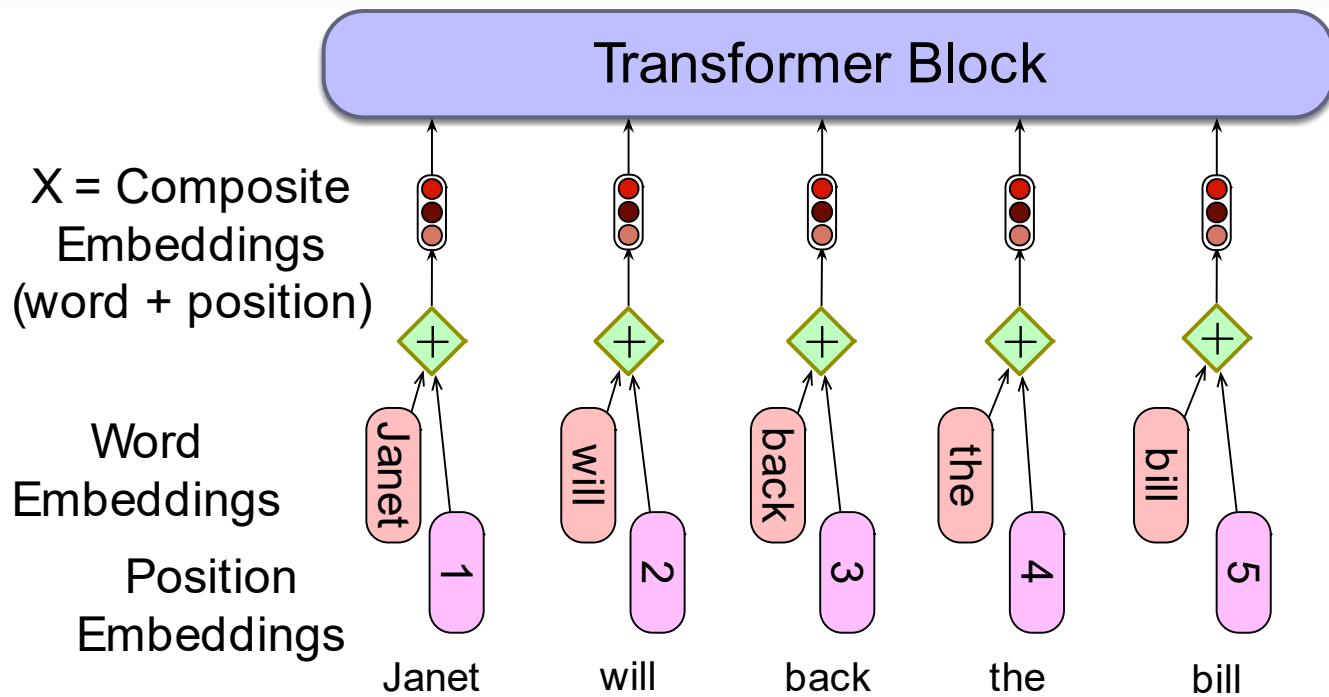
$w = [5, 4000, 10532, 2224]$

2. Select the corresponding rows from  $E$ , each row an embedding (row 5, row 4000, row 10532, row 2224).

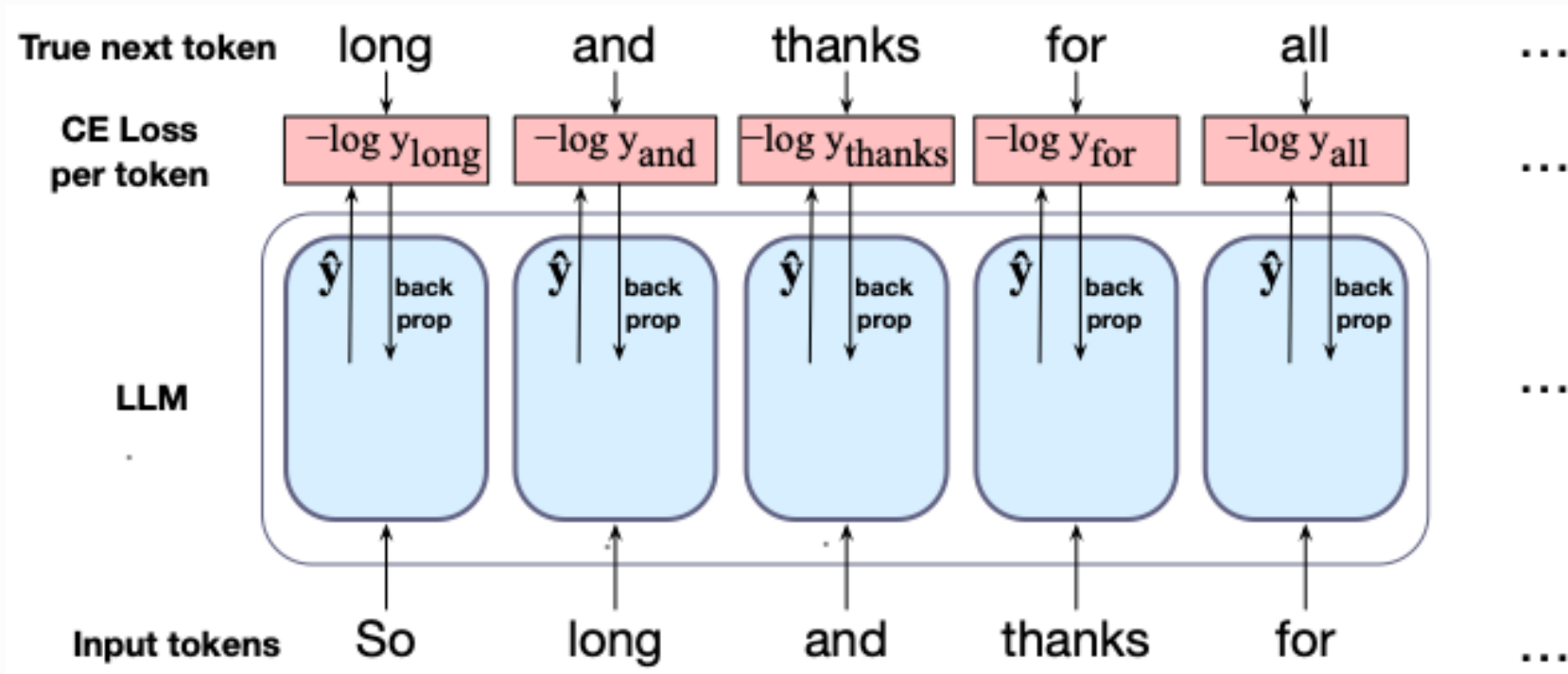
# Position Embeddings

- There are many methods, but we'll just describe the simplest: absolute position.
- Learn a position embedding matrix  $E_{\text{pos}}$  of shape  $d \times N$
- Start with randomly initialized embeddings
  - one for each integer up to some maximum length.
  - i.e., just as we have an embedding for the word *fish*, we'll have an embedding for position 3 and position 17.
- As with word embeddings, these position embeddings are learned along with other parameters during training.

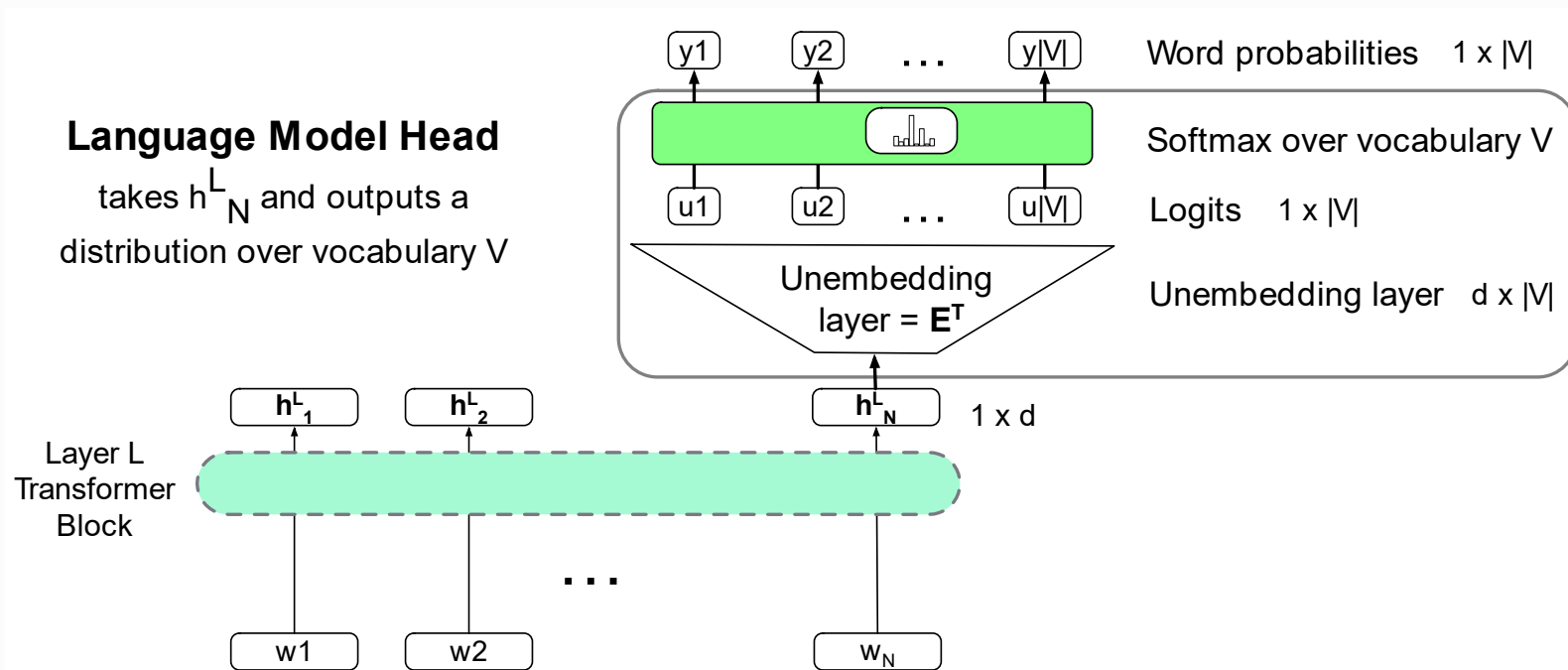
Each  $x$  is just the sum of token (word) and position embeddings



# Reminder: training a transformer language model

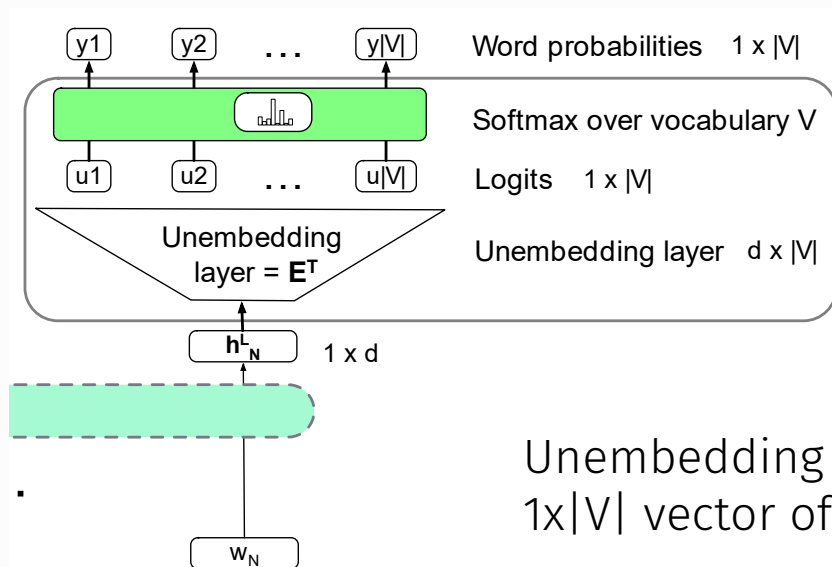


# Output: Language modeling head



# Language modeling head

**Unembedding layer:** FFN layer projects from  $h_N^L$  (shape  $1 \times d$ ) to probability distribution vector over the vocabulary

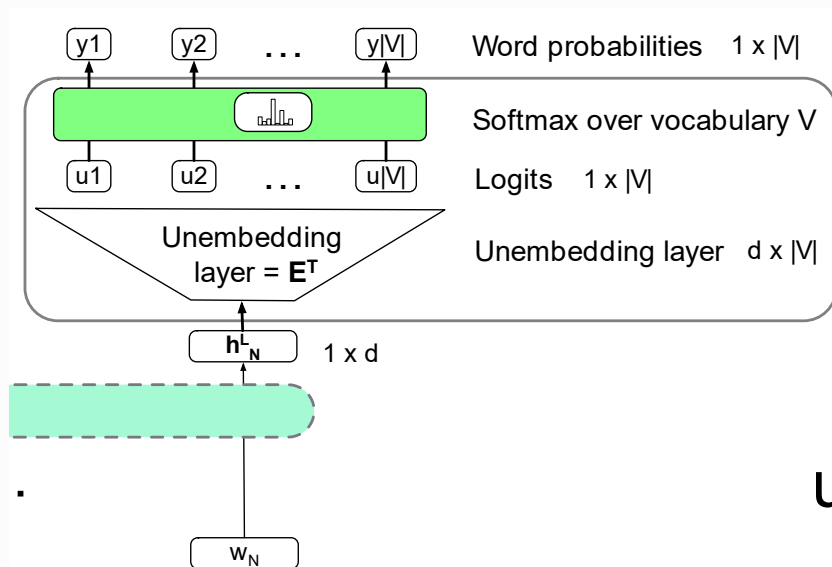


Why "unembedding"? **Tied** to  $E^T$

**Weight tying**, we use the same weights for two different matrices

Unembedding layer maps from an embedding to a  $1 \times |V|$  vector of logits

# Language modeling head



**Logits**, the score vector  $u$

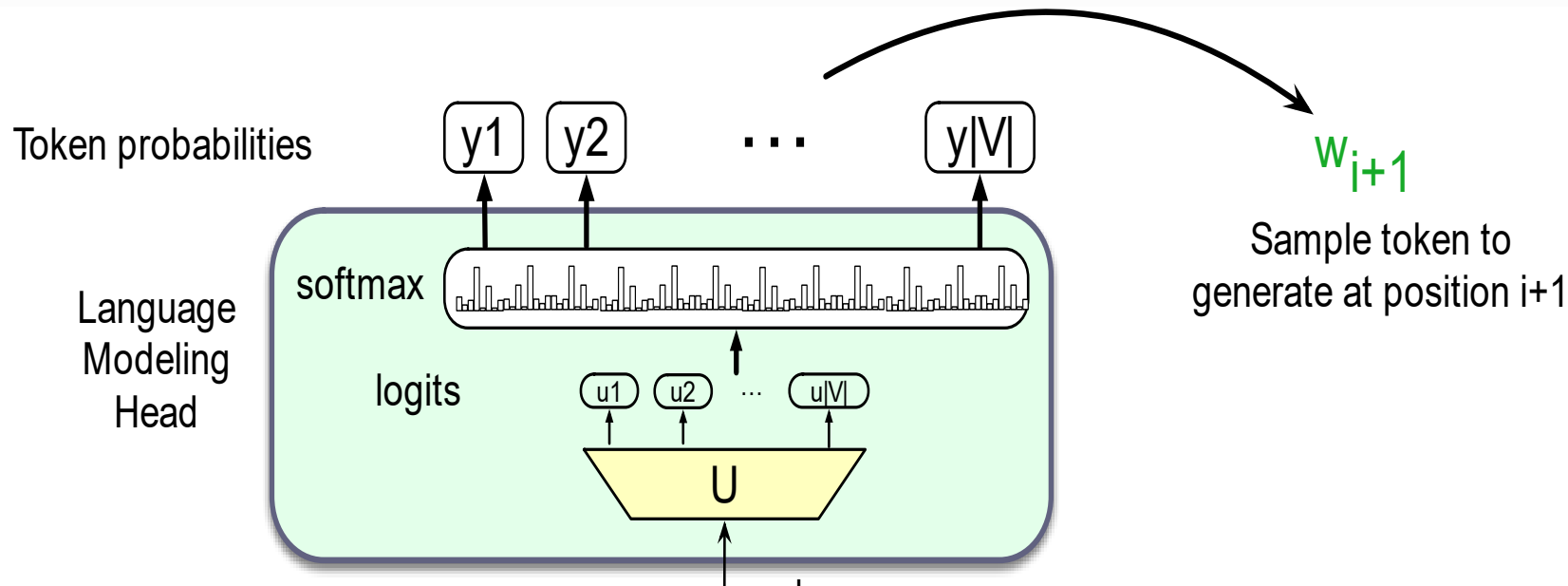
One score for each of the  $|V|$  possible words in the vocabulary  $V$ . Shape  $1 \times |V|$ .

**Softmax** turns the logits into probabilities over vocabulary. Shape  $1 \times |V|$ .

$$u = h_N^L E^T$$

$$y = \text{softmax}(u)$$

# The final transformer language model



## Activity: draw a diagram of a transformer-based LM

- Input all the way to predicting the next word
- Then explain its parts to a neighbor